What's cool in the new and updated OSGi Specs

Carsten Ziegeler

David Bosschaert
Speakers

Carsten Ziegeler (cziegeler@apache.org)
- RnD Adobe Research Switzerland
- OSGi CPEG and EEG Member
- ASF member

David Bosschaert (davidb@apache.org)
- RnD Adobe Research Dublin
- Co-chair OSGi EEG
- Open-source and cloud enthusiast
Agenda

- Framework updates
- Repository update
- OSGi/CDI integration
- Declarative Services
- Cloud
- Http Service
- Semantic Versioning Annotations
- Other spec updates
Framework Updates
Service Scopes (RFC 195)

- Service Scopes: singleton, bundle, prototype
- Driver: Support for EEG specs (EJB, CDI)
- Usage in other spec updates
- New PrototypeServiceFactory
public interface PrototypeServiceFactory<S> extends ServiceFactory<S> {

    S getService(Bundle bundle,
               ServiceRegistration<S> registration);

    void ungetService(Bundle bundle,
                      ServiceRegistration<S> registration,
                      S service);
}
New method in BundleContext:

```java
public interface BundleContext {

<S> ServiceObjects<S> getServiceObjects(ServiceReference<S> ref);

}
```

New Interface ServiceObjects:

```java
public interface ServiceObjects<S> {

    S getService();

    void ungetService(S service);

}
```
Repository 1.1
Client Bundle

Requirement req = 'capability 1'
matches = 
Repository.findProviders(req)

OSGi Repository

Resource 1
- capability 1
- capability 2
- capability 3

Resource 2
- capability 1
- capability 3

Client needs exporter

Requirement req = 'need exporter of org.foo version [2.1, 3)'
matches = 
Repository.findProviders(req)

OSGi Repository

Bundle A
- export
  org.foo; v2.1

Bundle B
- export
  org.foo; v2.1

OSGi Repository today
Example Repository namespaces

Client needs Blueprint

Requirement req = 'need osgi.blueprint extender version [1,2)'
matches = Repository.findProviders(req)

OSGi Repository
- Blueprint Impl 1
  - osgi.extender osgi.blueprint v1
- Blueprint Impl 2
  - osgi.extender osgi.blueprint v1

Client needs native driver

Requirement req = 'need provider of org.acme.driverx for Linux/x86-64'
matches = Repository.findProviders(req)

OSGi Repository
- DriverA.so
  - org.acme.driverx
  - os: Linux
  - processor: x86-64
- DriverB.dll
  - org.acme.driverx
  - os: Windows
  - Processor: x86
RFC 187 - Repository 1.1

Existing repository powerful

but: limited to queries in a single namespace

New in RFC 187:

- Combine requirements spanning multiple namespaces:

```java
Repository repo = ... // Obtain from Service Registry
Collection<Resource> res = repo.findProviders(
    repo.getExpressionCombiner().and(
        repo.newRequirementBuilder("osgi.wiring.package").
            addDirective("filter","(osgi.wiring.package=foo.pkg1)").buildExpression(),
        repo.newRequirementBuilder("osgi.identity").
            addDirective("filter",
                "(license=http://opensource.org/licenses/Apache-2.0)").buildExpression())
);```

RFC 193 - CDI Support
Bridging OSGi and standard JavaEE dependency model

- Publishing CDI beans as OSGi services
- Injecting OSGi services in CDI beans

CDI = Contexts and dependency injection
RFC 193 - Examples

● Publishing CDI bean as OSGi service

```java
import org.osgi.service.cdi.Component;

@Component
public class MyComponent implements MyInterface {
    ...
}
```

● Dependency Injection

```java
@Inject
@Service
private MyInterface serviceA;

@Inject
@Service(optional = true)
private MyInterface serviceB;
```
Declarative Services
RFC 190 - Declarative Services Enhancements

- Support of service scopes
- Diagnostic API
- DTOs
- But most importantly...
Use annotations for configuration...

```java
@interface MyConfig {
    boolean enabled() default true;
    String[] topic() default {"topicA", "topicB"};
    String userName();
    int service_ranking() default 15;
}
```
...and reference them in lifecycle methods

```java
@Component
public class MyComponent {

    String userName;

    String[] topics;

    @Activate
    protected void activate(final MyConfig config) {
        // note: annotation MyConfig used as interface
        if (config.enabled()) {
            this.userName = config.userName();
            this.topics = config.topic();
        }
    }
}
```
...or even simpler...

```java
@Component
public class MyComponent {

    private MyConfig configuration;

    @Activate
    protected void activate(final MyConfig config) {
        // note: annotation MyConfig used as interface
        if (config.enabled()) {
            this.configuration = config;
        }
    }
}
```
Annotation Mapping

- Fields registered as component properties
- Name mapping (_ -> .)
- Type mapping for configurations
@ObjectClassDefinition(label="My Component",
   description="Coolest component in the world.")
@interface MyConfig {
   @AttributeDefinition(label="Enabled",
      description="Topic and user name are used if enabled")
   boolean enabled() default true;

   @AttributeDefinition(...)  
   String[] topic() default {"topicA", "topicB"};

   @AttributeDefinition(...)  
   String userName();

   int service_ranking() default 15; // maps to service.ranking
RFC 190 - Declarative Services Enhancements

- Annotation configuration support
- Support of service scopes
- Introspection API
RFC 189 - Http Service Update

- Update to Servlet API 3+
- Whiteboard support
- and Introspection API
@Component(service = javax.servlet.Servlet.class,
    scope="PROTOTYPE",
    property={
        "osgi.http.whiteboard.servlet.pattern=/products/**",
    })

public class MyServlet extends HttpServlet {
    ...
}
Whiteboard Servlet Filter Registration

```java
@Component(service = javax.servlet.Filter.class,
    scope="PROTOTYPE",
    property={
        "osgi.http.whiteboard.filter.pattern=/products/*",
    })

public class MyFilter implements Filter {
    ...
}
```
Additional Support

- Most listener types are supported
  Register with their interface
- Error Pages and Resources
- Shared HttpContexts
- Target Http Service
Cloud
Current PaaS offerings...
OSGi *Cloud Ecosystems* PaaS
An OSGi cloud ecosystem...

- Many frameworks
  - hosting a variety of deployments
- Together providing *The Application*
- Not a bunch of replicas
  - rather a collection of different nodes
  - with different roles working together
  - some may be replicas
- Load varies over time
- ... and so does your cloud system
  - topology
  - configuration
  - number of nodes
  - depending on the demand
To realize this you need...

- **Information!**
  - need to know what nodes are available
  - ability to react to changes

- **Provisioning capability**

- **Remote invocation**
  - inside your cloud system
  - to get nodes to communicate
  - either directly...
  - ... or as a means to set up communication channels
RFC 183 - Cloud Ecosystems

FrameworkNodeStatus service:
- information about each Cloud node
- accessible as a Remote Service
- throughout the ecosystem

Information such as:
- Hostname/IP address
- Location (country etc)
- OSGi and Java version running
- A REST management URL
- Runtime metadata
  - Available memory / disk space
  - Load measurement

... you can add custom metadata too ...
<table>
<thead>
<tr>
<th>key</th>
<th>data type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.framework.uuid</td>
<td>String</td>
<td>The globally unique ID for this framework.</td>
</tr>
<tr>
<td>org.osgi.node.host</td>
<td>String+</td>
<td>The external host names or ip addresses for this OSGi Framework, if exists.</td>
</tr>
<tr>
<td>org.osgi.node.host.internal</td>
<td>String+</td>
<td>The internal host names or ip addresses for this OSGi Framework for access from inside the Ecosystem, if exists.</td>
</tr>
<tr>
<td>org.osgi.node.type</td>
<td>String</td>
<td>The name of the Cloud/Environment in which the Ecosystem operates.</td>
</tr>
<tr>
<td>org.osgi.node.version</td>
<td>String</td>
<td>The version of the Cloud/Environment in which the Ecosystem operates. The value follows the versioning scheme of the cloud provider and may therefore not comply with the OSGi versioning syntax.</td>
</tr>
<tr>
<td>org.osgi.node.country</td>
<td>String (3, optional)</td>
<td>ISO 3166-1 alpha-3 location where this Framework instance is running, if known.</td>
</tr>
<tr>
<td>org.osgi.node.location</td>
<td>String (optional)</td>
<td>ISO 3166-2 location where this framework instance is running, if known. This location is more detailed than the country code as it may contain province or territory.</td>
</tr>
<tr>
<td>org.osgi.node.region</td>
<td>String</td>
<td>Something smaller than a country and bigger than a location (e.g. us-east)</td>
</tr>
<tr>
<td>org.osgi.node.rest.url</td>
<td>String+ (URL, optional)</td>
<td>The external URL of the framework management REST API, if available.</td>
</tr>
<tr>
<td>org.osgi.node.rest.url.internal</td>
<td>String+ (URL, optional)</td>
<td>The ecosystem-internal URL of the framework management API, if available.</td>
</tr>
<tr>
<td>org.osgi.framework.version</td>
<td>String</td>
<td>The value of the Framework properties as obtained via BundleContext.getProperty().</td>
</tr>
<tr>
<td>org.osgi.framework.processor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>org.osgi.framework.os.name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>org.osgi.framework.os.version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.version, java.runtime.version, java.vm.vendor, java.vm.version, java.vm.name</td>
<td>String</td>
<td>The values of the corresponding Java system properties.</td>
</tr>
<tr>
<td>... additional properties ...</td>
<td></td>
<td>Additional properties may appear, set by the framework, Remote Services implementation or other entity.</td>
</tr>
<tr>
<td>... custom properties ...</td>
<td></td>
<td>See section 5.2.2.</td>
</tr>
</tbody>
</table>

**FrameworkNodeStatus service properties**
RFC 182 - REST API
A cloud-friendly remote management API works great with FrameworkNodeStatus

Example:

```java
addingService(ServiceReference<FrameworkNodeStatus> ref) {
    // A new Node became available
    String url = ref.getProperty("org.osgi.node.rest.url");
    RestClient rc = new RestClient(new URI(url));

    // Provision the new node
    rc.installBundle(...);
    rc.startBundle(...);
}
```
Additional ideas in RFC 183

- **Special Remote Services config type**
  - osgi.configtype.ecosystem
  - defines supported Remote Service data types
  - not visible outside of cloud system

- **Ability to intercept remote service calls**
  - can provide different service for each client
  - can do invocation counting (quotas, billing)

- **Providing remote services meta-data**
  - quota exceeded
  - payment needed
  - maintenance scheduled
Current OSGi cloud work

Provides a base line
  ○ to build fluid cloud systems
  ○ portability across clouds

Where everything is dynamic
  ○ nodes can be repurposed

... and you deal with your cloud nodes through OSGi services
Type and Package Annotations
Semantic Versioning...

... is a versioning policy for *exported packages*. 

OSGi versions: `<major>.<minor>.<micro>.<qualifier>`

Updating package versions:

- **fix/patch** (no change to API):
  - update micro

- **extend API** (affects implementers, not clients):
  - update minor

- **API breakage**:
  - update major

*Note*: not always used for *bundle* versions
RFC 197 – OSGi Type and Package Annotations

- Annotations for documenting semantic versioning information
  - Class retention annotations
- @Version
- @ProviderType
- @ConsumerType
Other Enterprise Spec updates
▷ Asynchronous Services
▷ Promises API
▷ Blueprint 1.1
  • Non-damped service references
  • Grace period enhancements
  • Many small fixes
▷ Remote Service Admin 1.1
  • Remote Service registration modification
▷ Subsystems 1.1
  • Provide Deployment Manifest separately
  • Many small enhancements
▷ DTOs
When can I get it?

RFCs available to everyone today:
https://github.com/osgi/design
(https://github.com/osgi/design)
Questions?